Writing A User Manual (part 1)

(2002-12-27) - Contributed by Deepa L, (c) Melonfire

Need to write a user manual, but don't know where to start? Our handy two-part guide takes you through the process, explaining the importance of proper planning in the early stages and demonstrating how to build a consistent and usable stylesheet for document formatting.

Depending on who you speak to, documentation is either the best part of a software project...or the worst.

Most developers wouldn't be caught dead writing a user manual - they much prefer spending their time building better, more efficient algorithms. Their users, on the other hand, don't really care about the code that powers a software application; they're more interested in getting their work done quickly, with minimal errors.

That's where support documentation, in the form of a user manual, comes in. Usually considered one of the least important deliverables, it is slowly coming of age, as software companies begin to realize the value of high-quality documentation that answers most user questions and reduces after-sales support calls (and expense).

Support documentation allows the user to use the delivered software with ease and efficiency. Ideally, it comprises:

- Interface text: The labels on interface elements like menu items, fields, instructions, confirmations, error messages et al.
- Application messages: Operational error messages and warnings.
- Online documentation: Online help, tutorial and searchable help pages.
- Print documentation: User manual and technical reference manual.

These, in totem, are the user's support system for usage of the software.

This article focuses on the user manual, explaining, from a technical writer's perspective, the process by which such a manual is developed, reviewed and delivered. I believe, though, that the process and planning tips are generic enough to apply to the other print documents, in accordance with both their purpose and scope.{mospagebreak title=Step By Step} One of the first decisions project managers face is whether the user manual should be a development team production, or whether a technical writer should be assigned the job. Practically, this decision is a function of the size and budget of the project, but an understanding of a technical writer's contribution helps in making an informed choice.

While the developer's responsibility is to be an expert on the structure, features and working of the software, the technical writer is responsible for understanding the users' mindset - their expectations, present level of expertise and possible questions - and then formulating the best way to communicate with them. The quality (ability to communicate) of the support documentation directly affects the level of after-sales support required. Since technical writers are trained in the black art of communication, it usually makes more sense to hire one for the support documentation needed, rather than have a developer do it.

Documentation, as a process, begins (ideally!) right at the point where the development team has finalized the software design, because:

- That provides enough fodder for you, the writer, to start planning the structure of the manual.
- You have a better chance of understanding the software if you ask

questions (lots of them!) while it's still in development.
Post-development, the team is usually into the next project, and you end up doing a lot of backtracking and guesswork.
- The project schedule will typically never allow time between development and delivery for anything more than very superficial documentation.
- Most important, the ideal scenario is to release the documentation in time for the software testing, so that the test team tests the documents as well as the software simultaneously.

An organized process of documentation will usually have the following phases:

- Planning
- Stylesheet creation
- Development
- Review
- Version management
- Delivery

In this article, I'll be focusing on the first two steps, with a list of things you should keep in mind when formulating the structure and style of your manual.{mospagebreak title=Asking The Hard Questions}
You should start thinking about the user manual right at the start, and try to have the following questions answered by the time you actually get down to writing it.

1. Who is the audience?

This helps you decide the tone and level of technicality of your language, the depth in which the concepts need to be explained, and (very important) the analogies that you can use (familiar ground is best when trying to explain something new). Knowing the following parameters about the intended users would help:

- What is their average age?
- Which computer software packages are they familiar with?
- What are the obstacles they usually experience while using these software applications?
- What are the top five task(s) they plan to use your software for?

- What is their current level of expertise (novice/intermediate/expert) in using particular software packages?

This information is useful when your software builds on existing software currently in use. For example, if you are delivering an intranet email utility that plugs in to Microsoft Outlook, it would make sense to find out if your audience has ever used it, and to what level.

This also brings up an important decision: do you decide the minimum technical expertise required of the users of your software, state it as such in the user manual, and get on with things? Or, given the results of your user profiling, do you take on the responsibility of bridging the gap between the current and required level of expertise (maybe by providing a short tutorial as a precursor to the manual)? The schedule and budget would normally make this decision for you.

The ideal scenario, of course, would be that you get all this information by interviewing the actual users. In case that isn't possible, your marketing and QA departments should have the requisite

insight into the target audience.

Besides this, some research into the business processes of the target organization will give you even greater insight into the context of user tasks, as well as fodder for analogies that may be easily understood y them. Additionally, customer meetings, including technical reviews, are great sources of audience information.

2. What is the scope of the document?

While the broad goals of the user manual would be to provide information on the installation, usage, administration and troubleshooting of the product, questions like these would help scope the document further:

- Current user expertise versus required expertise: What is the extent of background/explanation that needs to be given?
- Supported platforms: What are the different platforms/operating systems that the manual should address?
- Troubleshooting: What level of troubleshooting are the users supposed to handle? Is there a reporting mechanism for support? Is there separate documentation for troubleshooting?

3. What tool should you use for document development?

The user manual, online help and searchable help essentially build on the same information. Which means that your choice of tool, and its ability to allow you to reuse information from one document for the faster development of another, is crucial (especially if your project's on a tight schedule).

A number of good tools are available for this purpose. I like RoboHelp (http://www.ehelp.com/), though if you're working with XML, you should also look at the XMLMind XML Editor (http://www.xmlmind.com/)

4. What is the mode of document delivery?

The user manual can have two modes of delivery and distribution:

- Print: In this case, you take the responsibility of printing it in-house and delivering it to the customer (many customers demand this). The downside: you get to incur printing and distribution costs (and the accompanying logistical issues), together with recurring costs every time the documentation is revised.
- Electronic: In this case, you may choose to deliver documentation in electronic format, via CD-ROM at installation time, or provide downloadable material on your Web site. The de facto standard for such electronic documents is Adobe's Portable Document Format (PDF).

Again, if you're not sure what the final format will be, and if you're comfortable with XML, it's worthwhile considering developing your document in XML; this may then be easily converted into any other format at a subsequent stage.{mospagebreak title=Making Friends And Influencing People} Another important aspect of planning is figuring out your resource requirements, especially if you are a technical writer expressly brought in to the project for support documentation. There are a number of resources you can tap - here's a brief list:

1. SMEs: SMEs (Subject Matter Experts) are your guides throughout the documentation project. These are usually members of the development team who will familiarize you with the application, answer your questions and

generally be your information bank. This is a good time to determine which developers from the team are to be your SMEs.

Your relationship with the SMEs will go a long way in determining the success of this task.

- Determine a method of communication that is suitable to both. An option is that you post your questions to the SMEs via e-mail, who may then respond in their spare time, or (if the explanation is long-drawn) schedule a meeting.
- Ask the right questions. Understand that on the other side of your question lies a lot of information, and what you get to know will be in direct response to only what you ask. So, spend some time getting your questions right.
- Get familiar with the platform and terminology used in the software. This way, again, you make your meetings with the SMEs efficient.
- Let the SMEs know that you need to know of every change made in the project; any change in the software that affects flow, functionality or interface affects your document. In fact, even with changes that don't affect the user interface, it's a good idea to be in the loop, because there could be reactions that you would want to know about. Again, try and set up an information chain or e-mail trigger for the same.

2. Project specifications: Needless to say, getting acquainted with the specification documentation is crucial to understanding the project. The objective of the project from the customer's business point of view is usually defined very clearly in these - make sure you re-use that, as your users will relate to it.

3. Prototype: Since you're going to be writing about the behavior of each feature in the software, playing around with the actual interface is a must. On the other hand, documentation usually begins in parallel with development, so you don't really have anything to go by.

The workaround here is the prototype. The delivery of the prototype by the development team will be a big milestone in your schedule...because that's where you actually start developing the manual. Get this date from the developers, and circle it in your calendar.

Note also that changes take place frequently in the early stages of development, not only in the behavior of the software, but also in the interface elements, text labels and messages. Ensure that your manual reflects the delivered product by referring to the latest prototype.

4. Schedule: The cornerstone of this planning stage is the schedule. An important consideration here is the dependencies between your tasks and other milestones in the schedule. Understand the developers' schedule and build your own based on that. Your milestones could be something like this:

- User profile generated
- Product information assimilated from specifications
- Stylesheet finalized
- Table of contents/outline complete
- Outline sent for review
- Outline returned with comments
- Comments incorporated and outline available for sign-off
- Sign-off
- First draft sent for review
- First draft returned with comments
- Comments incorporated and draft available for sign-off
- Sign-off

- Second draft sent for review
- Second draft returned with comments
- Comments incorporated and draft available for sign-off
- Sign-off
- Third draft sent for review
- Third draft returned with comments
- Comments incorporated and draft available for sign-off
- Sign-off
- Delivery

Review and revision efficiency (addressed in the second part of this article) are crucial to ensuring that three drafts are all it takes.{mospagebreak title=Being Conventional} Conventions in the document lead to patterns that the users can grasp. They then start expecting information in a particular format, thus increasing their level of comfort with the document. Using consistent styles also speeds up assimilation of the information, and helps spot particular information easily on re-reads.

- Headings: Headings are a powerful tool in making a huge mass of text look manageable. A common model is that as you go deeper in a particular topic, you indicate that by descending prominence of headings. So, all top level headings will be, say, in a large font size and bold typeface, with the next level taking a smaller font size, and so on. You might also want to number the headings to help users understand the grouping of information.
- Styles: While a short piece of text requires only minimal use of styles (bold for highlighting, underline for warning), a tome as voluminous as a user manual needs you to be much more creative. You could set conventions for indicating screen names, interface text or text that the user needs to input.

On the other hand, too many conventions negate the purpose - remember, they should assist in quick reading and lookup, and they won't if users have to keep recollecting what a particular style indicates.

- Indented text and footnotes: This is text that is peripheral to the point that you are making - for example, background information on a concept that you're introducing or a warning related to some functional step that you're explaining. Add these when you don't want to distract the user from the main flow of information.
- Bullets and numbering: Bullets and numbering can also help to break up complex concepts into simpler, smaller information nuggets. The convention here is to use numbering for sequential information only (for example, steps to perform a task) and bullets for other related information that is best presented in points instead of a paragraph.

Bullets also allow you to group together points related to a concept and ascribe them levels of importance. Much care and consideration should be given to the grouping of information in this manner - it could easily be as confounding as useful.
- Terminology: A very, very important rule of creating end-user documentation is to be consistent in your use of words. For example, if you're using the word "function" to indicate the, well, functions of your software, you shouldn't at any point switch to "features", "commands", "menu items" or "actions". To this end, make yourself a glossary of the terms that you're going to use right at the start, and stick to them consistently.
- Images and illustrations: Sometimes, a picture really is worth a thousand words - and screen grabs, schematics or flow diagrams can substantially increase the efficacy of your document. Plan your usage of

images and illustrations in advance, and be consistent in their usage and labeling.

So that takes care of preparation - all that's left now is to actually begin work. In the next, and concluding, section, I'll be discussing the process of actually developing the structure of the manual, together with a sample table of contents, and also spending some time on document revisions, version management, and delivery. Make sure you come back for that!

Note: Examples are illustrative only, and are not meant for a production environment. Melonfire provides no warranties or support for the source code described in this article. YMMV!

Practices

Writing A User Manual (part 2)
Contributed by Deepa L, (c) Melonfire
2003−01−03

The **Slingbox AV** is the perfect companion for your DVR, digital cable, or satellite receiver. It allows you to watch and control live TV, or any other TV entertainment you have, anywhere you go. **Save $10**!

In the first part of this article, I introduced you to the process of creating a user manual, illustrating the groundwork you need to do prior to actually beginning work. In this concluding article, I'll take you through the development process, demonstrating a sample TOC and giving you a few tips on the review and version management stages of the process.

Once you've got your stylesheet done, you've finally reached the point of production − where you actually start creating the document. Generally, the best way to go about it is to create the bare−bones structure (TOC), have it approved, fill in the peripheral information (overview, scope, conventions used, glossary) and then move in to the meat of the matter. This is also the time (ideally!) when the prototypes are finally ready for you to start working with.

The following is a generic user manual structure:

1. Introduction
    - ♦ The product − introduce the product to the user.
    - ♦ The user manual
        - ◊ Scope/Purpose
        - ◊ Flow
        - ◊ Conventions
        - ◊ Glossary
2. Installing the software (assuming it's not a separate guide)
    - ♦ System requirements
        - ◊ Platform Support
    - ♦ Information/resources required in the process of installation
    - ♦ Installation steps

At this point there is usually a decision to be made about how to depict installation procedures for different platforms. The main criteriion here is how different the procedures are − if the steps are drastically different, you will have to explain the procedure separately for each platform. But if the steps are not very different, you could choose the most common platform as your base, and wherever the steps are different, indicate the steps for the different platforms as indented text.

3. Using the software
    - ♦ Introduction
        - ◊ Purpose of the software
        - ◊ What it does and does not do (list the exact tasks)
        - ◊ User levels and the implications (segregate the user and admin level tasks)
    - ♦ Best configuration (for example, best viewed in 640x480 resolution)
    - ♦ Invoking the software
    - ♦ Interface elements
    - ♦ Steps to perform the required tasks
4. Administration
    - ♦ Reiterate the administration level tasks
    - ♦ Segregate (if possible) into administration, maintenance and troubleshooting functions, and then get into

explanations
- Always lead in to a task with scenarios (for example, you need to shut down the server over the weekends and at the end of the day − here's how...)
- Also, try and bring out exceptional scenarios at the same time (to continue the above example, the administrator would not shut down the server over the weekend if there has been a request for remote access by one of the users)

5. Troubleshooting
   - For each error condition describe:
     - What happens on the display
     - The error message displayed
     - What it means and what is the implication with respect to the attempted action (for example, the user will have to re−enter information)
     - Steps to take to rectify the error
6. Appendix
   An appendix allows you to expound on peripheral information that would be detracting when given in the main body. Detailed diagrams, flow charts, or references to books/tutorials on related software could be included here.

Here are some quick tips to assist you in developing this structure:

- Be visual: The most comforting thing for the user will be to see on screen what they've seen on the manual's pages, or vice−versa. Try and use screen grabs and small schematic diagrams wherever appropriate.
- Importance of relevant analogies: Essential if your software introduces concepts new to the user.
- Use of transition words: "because", "therefore" and "consequently" are powerful words when talking about cause−effect relationships that the user isn't aware of.

Once you're done with filling in the details, it's time for a review. The review is a crucial process for any document, and more so for the user manual. You are dependent on the SMEs for verification of the technical information, on peer writers for editorial comments on structure and flow, technical support guys for evaluation of whether you've covered the most common support requests, and − of course! − a sample set of actual users to tell you whether it works.

But before you get to the point where you release drafts for review by these groups, you need to review the document yourself. Here's a quick cheat sheet that assists in this process:

1. Is all the information there? Have all the key terms been defined? Are instructions to cover all tasks there?
2. Is the usage of paragraphs appropriate? Are they too long? On the other hand, guard against the paragraphs being so short and abrupt that the information seems unrelated.
3. Does the flow make sense?
4. Are the levels logical? Is the grouping together of modules sensible?
5. Are the descriptive headings apt? Could there be a summary at the beginning and end of each section to guide the user better?
6. Are there inconsistencies in usage of terms?
7. Is the tone consistent throughout? Are you shifting from the first to the third person and back again?
8. Are all topics covered to a consistent level of depth? Sometimes, at the time of writing, you may not have the same level of information on all topics covered (or you may just have been in a hurry to get the job done). This difference is very noticeable, and looks sloppy − watch out for it and fill in the consistent level of detail.

Once you've covered this ground, you can release the draft for review by the others involved. A thorough review needs a lot of commitment and time. You will need to make it easier for your reviewers by including quick cheat sheets stating what they should be looking for. An example cheat sheet for the SMEs would be:

1. A technical review is not an editorial review.

2. Focus on the technical facts to verify that the technology works as documented.
3. Verify the technical accuracy of all procedural steps included in the document.
4. Verify the technical accuracy of all screen captures in the document.

User reviews are a tad trickier than the others are because of the lack of resources. First, you may not have access to the actual users to review your document. And second, they may not really be motivated at that point to take the time to review your document. The workaround is to use your marketing and QA departments, and perhaps the people from the customer's end who are involved in the project.

Once the reviews are in, you need to get down to implementing the changes suggested. One tip would be to start revision on a document only after all the review comments are in. Also, while we won't get into the art of accepting feedback, you have to be in control of the changes that you agree to make. While you can change the information quite a bit at the time of the first review, you should try and restrict your changes to corrections only after the second review; structural changes this late in the process will throw you off. A characteristic of documentation is that if you notice even one inaccuracy in a document, it will put you off going through the rest of it. The gravity of this increases manifold when you're talking about a user who's looking to this document to understand your software. Ensuring that your manual reflects the latest version of the software is crucial, and this is where tying the document version number with that of the software comes in.

Another consideration here is version nomenclature. You could tie this in with the software, using x.y nomenclature that has x changing with every baseline change and y changing for every intermediate release of the document. Also, when you revise the document, you should record the reason/description of the change in the document's revision log. With all of this behind you, you will finally be ready to release the manual. The following frills will complete the package:

1. Cover page
    - ♦ The name of the software should be written in accordance with the brand decided.
    - ♦ The version number of the software should be clearly stated.
    - ♦ The name of the developer with address and contact numbers.
2. Table of contents
    - ♦ The topics should be linked to the matter inside.
3. Notifications for proprietorship and confidentiality
4. Headers and footers
    - ♦ Headers could include the project name and version number of the document.
    - ♦ Footers can have the page numbers and a short confidentiality notice.

It might also be a good idea to include a feedback form as the last page, as your users will probably get back to you with suggestions. This will be especially useful if there is a second phase of development for the software.

And that's about it. If you'd like to learn more about the technical writing process in general, here are some links you might find useful.

Developing documentation without a tech writer, at
http://builder.com.com/article.jhtml?id=u00320020510gcn02.htm&vf=ra

The Seven Deadly Assumptions of Technical Communication, at
http://www.williamrice.com/techcomm/sevenassumptions/7assump1.html#Assum ption1

Online Technical Writing − An Online Textbook at http://www.io.com/~hcexres/tcm1603/acchtml/acctoc.html

Happy writing!

Note: Examples are illustrative only, and are not meant for a production environment. Melonfire provides no warranties or

support for the source code described in this article. YMMV!

DISCLAIMER: The content provided in this article is not warranted or guaranteed by Developer Shed, Inc. The content provided is intended for entertainment and/or educational purposes in order to introduce to the reader key ideas, concepts, and/or product reviews. As such it is incumbent upon the reader to employ real−world tactics for security and implementation of best practices. We are not liable for any negative consequences that may result from implementing any information covered in our articles or tutorials. If this is a hardware review, it is not recommended to open and/or modify your hardware.